Software Testing and Validation 2016/17
Instituto Superior Técnico

Project

Due: April 6, 2018

1 Introduction

The development of large information systems is a complex process and demands several layers of abstraction. One first step is the specification of the problem in a rigorous way. After a rigorous specification of the problem, one may perform formal analysis and prove correctness of the implementation. If the implementation is not correct, one may discover flaws that would not be easy to detect in any other way. Complementarily, and in particular when a solution has already been implemented, the quality of the solution may be assessed by running tests in order to detect existing flaws.

The project of this course consists in the design of a set of test suites given a specification. This specification models an application that manages the data of a provider of mobile telephony called *Vos*. The main goal of this project is for the students to learn the concepts related to software testing and acquire some experience in designing test cases.

Section 2 describes some entities present in the system to test and some implementation details concerning these entities that are important for testing these entities. Section 3 describes the test cases to implement and how the project will be evaluated.

2 System Description

The system to test is responsible for the communication and client management of the provider of mobile telephony called *Vos*. This system manages the clients of *Vos* and the phone numbers, mobile phones and telecommunications that concern clients of *Vos*. In this sytem, each client has a bounded number of phone number belonging to *Vos* Each phone number of a client of *Vos* can be associated with a given mobile phone. The system has to keep the information concerning all communications made by each mobile phone belonging to the clients of *Vos* so that it is able to compute the communication cost for each client.

2.1 The ClientManager entity

The `ClientManager` entity is responsible for managing the clients and the phone numbers of *Vos*. This entity keeps the registered clients and knows all phone numbers *Vos* (those already assigned and those that are still free). Each phone number can be assigned at most to a single client. In this context, a valid phone number is an integer with 9 digits. Figure 1 shows the interface of this entity. In the context of this entity, each client is identified by his social security number (designated as nif).

The method `assignPhoneNumber` assigns a free phone number to a client of *Vos* if all conditions are met. If at least one of these conditions does not hold, then this method does not change anything and throws the `InvalidOperationException` exception.

The responsibility of `computeBil` method is to determine the value to pay for a client taking into account all communications made by the client through all of his registered mobile phones. This value should be computed as follows:

```
public class  ClientManager {

  // Creates a new client of Vos and stores it. If phoneNumber is a number
  // already assigned or it is not valid, then InvalidOperationException is
  // thrown. If there is already a registered client with an identification
  // number equal to nif, then the DuplicateNIFException is thrown.
  public Client register(String clientName, int nif, int phoneNumber)
       throws DuplicateNIFException, InvalidOperationException { /* .... */ }

  // returns the client with the desired nif, null if not found
  public Client getClient(int nif) { /* .... */ }

  // Assigns a free phone number of Vos to the client of Vos identified by nif. If it is not
  // possible to assign the phone number, then the InvalidOperationException is thrown.
  public final void assignPhoneNumber(int clientNif, int phoneNumber) throws InvalidOperationException
   { /* .... */ }

  // computes the bill of the client taking into account his communications.
  public float computeBill(Client client) { /* .... */ }

  // returns the list of clients of Vos
  public List<Client> getClients()  { /* .... */ }

  ...
}
```

Figura 1: The interface of*ClientManager* class.

- If the number of calls made is less than 10, then the cost of each call is $0,15$ and the cost of each SMS is $0,05$ if the number of SMSs sent is less than 300 or $0,045$ otherwise.
- If the number of calls made is greater than or equal to 10 but less than 50, then the cost of each call is $0,10$. In this case, the cost of each SMS depends on the number of SMSs sent. If the client has sent less than 100 SMS, then the cost of each SMS is $0,04$, otherwise, the cost of each SMS is $0,035$.
- If the number of calls made is greater than or equal to 50 but less than 500, then the cost of each SMS and call depends on the number of SMSs sent. If this number is lower than 300, then the cost of each call and SMS is $0,09$ and $0,03$, respectively. Otherwise, the costs are $0,08$ and $0,02$, respectively.
- Finally, if the number of calls is greater than or equal to 500, the cost of each call is $0,06$ and SMSs are free

2.2 The Client entity

Each client of *Vos* has a name (a string of characters with a minimal size equal to 5) and by its social security number (designated as nif). This number is a unique identifier in *Vos*. A client can have several phone numbers managed by *Vos* (between 1 and 5). Each client can associate a mobile phone to each of his assigned phone numbers. Figure 2 shows the interface of this entity.

Each client can register in the system a given amount of phone number of *friends*. The maximum number of phone number a client can register is equal to three times the number of phone numbers plus five. The phone numbers registered as friends have some privileges which are described in Section 2.3).

2.3 The Mobile entity

).

In the context of the system to implement a mobile phone can make and receive calls and send and receive SMSs. A mobile phone can be turned off (and in this case it cannot make calls, send SMS and receive calls or SMSs) or turned on. Figure 3 shows the interface of this entity.

```
public class  Client {
  Client(String name, int nif, int phoneNumber) { /* .... */ }

  // Management of the phone numbers of this client
  void addPhoneNumber(int phoneNumber) throws InvalidOperationException { /* .... */ }
  void removePhoneNumber(int phoneNumber) throws InvalidOperationException  { /* .... */ }

  // registers a phoneNumber of this client to a given type of mobile phone.
  public Mobile registerMobile(int phoneNumber) throws InvalidOperationException { /* .... */ }

  // unregisters the mobile phone associated with phoneNumber
  public void unregisterMobile(int phoneNumber) throws InvalidOperationException { /* .... */ }

  // returns the mobiles of this client
  public List<Mobile> getMobiles() { /* .... */ }

  // returns the phone numbers assigned to this client.
  public List<Integer> getPhoneNumbers() { /* .... */ }

  // returns name of client
  public String getName() { /* .... */ }

  // return list of registered friends
  publi List<Integer> getFriends() { /* .... */ }

  // friends management
  public void addFriend(int phoneNumber) throws InvalidNumberPhoneException { /* .... */ }
  public void removeFriend(int phoneNumber) throws InvalidOperationException { /* .... */ }
  ...
}
```

Figura 2: A interface da classe *Client*.

A mobile phone can be turned on or off. A turned on mobile phone may make/receive calls and send/receive SMSs and it has two modes (*friend* and *silent*) that can be enabled or disabled. Concerning the invocation of the methods of this entity, we have the following restrictions:

- The mobile phone can be turned on or off. If it is turned off, then it is possible to turn it on (through invocation of `turnOn` method).
- When a mobile phone is turned on both modes are disabled. A turned on mobile phone with neither mode active can make and receive calls (through `makeCall` and `acceptCall` methods) and send and receive SMSs (through `sendSMS` and `receiveSMS` methods).
- At any time, it is possible to change both modes of a turned on mobile phone through the `setFriendMode` and `setSilentMode` . If a mobile phone only has the *silent* mode enabled, then it can just send and receive SMSs. If it only has the *friend* mode enabled, then it can make calls, send SMSs but it can only receive calls and SMSs from numbers registered as friends of the client associated with the concerned mobile phone. Finally, if the mobile phone has both modes active, then it can send SMSs and only receive SMSs from phone numbers registered as *friends*.
- A turned on mobile phone can be turned off (through `turnOff` method) at any time.
- At any time, it is possible to know the status of a mobile phone (if the mobile phone is on or off and which modes are enabled) using the `getStatus` method.

When a mobile phone is created it is turned off. You should also consider that any invocation of a method of this entity in a case not described before corresponds to an invalid invocation and should lead to throwing the *InvalidOperationException* exception.

```
public class  Mobile {
  // Creates a mobile with the specified phone number associated with the specified client.
  public Mobile(Client client, int phoneNumber) { /* .... */ }

  // Turns on the mobile with both modes disabled.
  public void turnOn() throwns InvalidOperationException { /* .... */ }

  // Turns off the mobile.
  public void turnOff() throwns InvalidOperationException { /* .... */ }

  // Changes the silent mode of a turned on mobile.
  public void setSilentMode(boolean silentMode) throwns InvalidOperationException { /* .... */ }

  // Changes the friend mode of a turned on mobile.
  public void setFriendMode(boolean friendMode) throwns InvalidOperationException { /* .... */ }

  // Sends an SMS to the specified phone number,
  void sendSMS(String smsMessage, int calledPhoneNumber) throws InvalidPhoneNumberException,
     InvalidOperationException { /* .... */ }

  // Receives an SMS from the specified phone number
  void receiveSMS(String smsMessage, int callerPhoneNumber) throwns InvalidOperationException { /* .... */ }

  // Makes a call to the specified phone number.
  void makeCall(int calledPhoneNumber) throws InvalidPhoneNumberException, InvalidOperationException
  { /* .... */ }

  // Accepts a phone call from callerPhoneNumber
  void acceptCall(int callerPhoneNumber) throwns InvalidOperationException { /* .... */ }

  // Returns a string describing the status of this mobile phone: turned off/on and silent/friend mode
  public String getStatus() { /* .... */ }

  ....
}
```

Figura 3: A interface da classe *Mobile*.

3 Project Evaluation

Students should develop the required test cases applying the most appropriate test patterns. The test cases to design are the following:

- Test classes *Client* and *Mobile* at class scope.
- Test methods *assignPhonenumber* of class *ClientManager* and *computeBill* of class *ClientManager*.
- Finally, it is necessary to implement six test cases of the test suite that exercises the class *Client* at class scope. This implementation should use the *TestNG* testing framework.

The implemented test cases will be evaluated as follows:

1. Development of the test cases for class *Client*: 0 to 3.5.
2. Development of the test cases for class *Mobile*: 0 to 6.5.
3. Development of the test cases for method *assignPhonenumber*: 0 to 3.5.
4. Development of the test cases for method *computeBill*: 0 to 3.5.
5. Implementation of six test cases concerning the test suite that tests class *Client* using the *TestNG* framework: 0 to 3.0.

For each method or class under testing, it is necessary to indicate the following:

- The name of used test pattern.
- If applicable, the result of the different stages of the application of the test pattern using the format described in the lectures.
- Description of the test cases that result from the application of the chosen test pattern.

If one of the above items is only partially developed, the grade will be given accordingly.

3.1 Other Forms of Evaluation

It may be possible a posteriori to ask the students to individually develop test cases similar to the ones of the project. This decision is solely taken by the professors of this course. Students whose grade in the first test is lower than this project grade by more than 5 will have to make an oral examination. In this case, the final grade for the project will be individual and will be the one obtained in this evaluation.

3.2 Fraud and Plagiarism

The submission of the project presupposes the commitment of honor that the project was solely executed by the members of the group that are referenced in the files/documents submitted for evaluation. The forfeit of this commitment, i.e., the appropriation of work done by other groups, will have as consequence the exclusion of the involved students (including those that allowed the appropriation) from this course.

3.3 Handing-in the Project

The project is due April 6th, 2018 at 17:00. The protocol for handing in the project will be available soon in section *Project* of the webpage of the course.

4 Final Remarks

All information regarding this project will be available in section *Project* of the webpage of the course. Extra material such as links, manuals, and FAQs will also be available in that same section.

Have a Good Work!